Android Studio - Zegar analogowy

Ogólne

Aplikacja prezentuje dynamiczny zegar analogowy wskazujący aktualny czas.

Uwaga: Nigdy nie wolno kopiować kodu z PDF-a, gdyż zawiera on niewidoczne znaki końca linii, twarde odstępy, odmienne cudzysłowy, odmienne apostrofy, odmienne myślniki. To godziny dodatkowej pracy na wykrywanie błędów i ich poprawianie. Kod należy przepisać ze zrozumieniem.

Kod - wersja podstawowa

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ZegarAnalogowyTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    ZegarAnalogowy()
                }
            }
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ZegarAnalogowy() {
    Column {
        CenterAlignedTopAppBar(
            title = { Text("Zegar dynamiczny", fontWeight = FontWeight.ExtraBold) },
            colors = TopAppBarDefaults.topAppBarColors(
                containerColor = Color(47, 79, 79, 255),
                titleContentColor = Color.White
            )
        )

        Spacer(modifier = Modifier.height(30.dp))
        var aktualnyCzas by remember { mutableStateOf(Calendar.getInstance()) }
        LaunchedEffect(Unit) {
            while (true) {
                aktualnyCzas = Calendar.getInstance()
```

```
        delay(1000)
    }
}

Box(
    modifier = Modifier.fillMaxSize(),
    contentAlignment = Alignment.Center
) {
    Canvas(modifier = Modifier.size(280.dp)) {
        val centerX = size.width / 2
        val centerY = size.height / 2
        val promienTarczy = min(centerX, centerY) + 2
        rotate(degrees = -90f, pivot = Offset.Zero) {
            translate(left = -size.height, top = 0f) {
                drawCircle(
                    color = Color.White,
                    radius = promienTarczy,
                    center = Offset(centerX, centerY)
                )
            }
        }

        repeat(12) { hour ->
            val indeksyKat = Math.PI / 6 * hour
            val poczatekIndeksuX = centerX + cos(indeksyKat) * (promienTarczy - 54)
            val poczatekIndeksuY = centerY + sin(indeksyKat) * (promienTarczy - 54)
            val koniecIndeksuX = centerX + cos(indeksyKat) * (promienTarczy - 12)
            val koniecIndeksuY = centerY + sin(indeksyKat) * (promienTarczy - 12)
            rotate(degrees = -90f, pivot = Offset.Zero) {
                translate(left = -size.height, top = 0f) {
                    drawIntoCanvas { canvas ->
                        val rysik = Paint().asFrameworkPaint()
                        canvas.nativeCanvas.drawLine(
                            poczatekIndeksuX.toFloat(),
                            poczatekIndeksuY.toFloat(),
                            koniecIndeksuX.toFloat(),
                            koniecIndeksuY.toFloat(),
                            rysik.apply {
                                color = Color.Black.toArgb()
                                strokeWidth = 6.dp.toPx()
                            }
                        )
                    }
                }
            }
        }
```

```
            val godzina = (aktualnyCzas.get(Calendar.HOUR_OF_DAY) % 12)
            val minuta = aktualnyCzas.get(Calendar.MINUTE)
            val sekunda = aktualnyCzas.get(Calendar.SECOND)
            val pozycjaKatowaGodziny = Math.PI / 6 * (godzina + minuta / 60.0)
            val pozycjaKatowaMinuty = Math.PI / 30 * (minuta + sekunda / 60.0)
            val pozycjaKatowaSekundy = Math.PI / 30 * sekunda
            // WSKAZÓWKA GODZINOWA:
            rysujWskazowke(
                centerX,
                centerY,
                promienTarczy * 0.5f,
                pozycjaKatowaGodziny,
                10.dp,
                Color.Black
            )
            // WSKAZÓWKA MINUTOWA:
            rysujWskazowke(
                centerX,
                centerY,
                promienTarczy * 0.86f,
                pozycjaKatowaMinuty,
                8.dp,
                Color.Black
            )
            // SEKUNDNIK:
            rysujWskazowke(
                centerX,
                centerY,
                promienTarczy * 0.95f,
                pozycjaKatowaSekundy,
                2.dp,
                Color.Red
            )
        }
    }
}
}

fun DrawScope.rysujWskazowke(
    przekazaneCenterX: Float,
    przekazaneCenterY: Float,
    przekazanaDlugoscWskazowki: Float,
    przekazanaPozycjaKatowaWskazowki: Double,
    przekazanaSzerokoscWskazowki: Dp,
    przekazanyKolorWskazowki: Color
) {
    val koniecWskazowkiX = przekazaneCenterX + cos(przekazanaPozycjaKatowaWskazowki) *
        przekazanaDlugoscWskazowki
```

```
    val koniecWskazowkiY = przekazaneCenterY + sin(przekazanaPozycjaKatowaWskazowki) *
        przekazanaDlugoscWskazowki

    rotate(degrees = -90f, pivot = Offset.Zero) {
        translate(left = -size.height, top = 0f) {
            drawIntoCanvas { canvas ->
                val rysik = Paint().asFrameworkPaint()
                rysik.color = przekazanyKolorWskazowki.toArgb()
                canvas.nativeCanvas.drawLine(
                    przekazaneCenterX,
                    przekazaneCenterY,
                    koniecWskazowkiX.toFloat(),
                    koniecWskazowkiY.toFloat(),
                    rysik.apply {
                        strokeWidth = przekazanaSzerokoscWskazowki.toPx()
                    }
                )
            }
        }
    }
}
```

Kod - wersja rozbudowana

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            ZegarAnalogowyTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    ZegarAnalogowy()
                }
            }
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ZegarAnalogowy() {
    Column {
        CenterAlignedTopAppBar(
            title = { Text("Zegar dynamiczny", fontWeight = FontWeight.ExtraBold) },
            colors = TopAppBarDefaults.topAppBarColors(
```

```
            containerColor = Color(47, 79, 79, 255),
            titleContentColor = Color.White
        )
    )


// Kod Zegara Analogowego:
    Spacer(modifier = Modifier.height(30.dp))
    var aktualnyCzas by remember { mutableStateOf(Calendar.getInstance()) }
    // Operacje asynchroniczne, wykonywane poza wątkiem głównym aplikacji,
    // gdyż mogą potrzebować więcej czasu (np. pobieranie informacji z Internetu).
    // Są wątkiem pobocznym (oderwanym, niezależnym, w tle) od głównego wątku:
    LaunchedEffect(Unit) {
        // Chcemy tu mieć wieczną pętlę (odczytuje bieżący czas co sekundę):
        while (true) {
            aktualnyCzas = Calendar.getInstance()
            delay(1000)
        }
    }

    Box(
        modifier = Modifier.fillMaxSize(),
        contentAlignment = Alignment.Center
    ) {
        Canvas(modifier = Modifier.size(280.dp)) {
            val centerX = size.width / 2 // W pikselach
            val centerY = size.height / 2 // W pikselach
            val promienTarczy = min(centerX, centerY) + 2 // Wielkość tarczy, promień
            // Obrót tarczy zegarka o 90 stopni w lewo (zawsze tuż przed użyciem funkcji draw...).
            // Atrybut pivot oznacza punkt obrotu.
            rotate(degrees = -90f, pivot = Offset.Zero) {
                // Przesunięcie punktów po obróceniu
                translate(left = -size.height, top = 0f) {
                    // Rysowanie tarczy zegarka:
                    drawCircle(
                        color = Color.White,
                        radius = promienTarczy,
                        center = Offset(centerX, centerY)
                    )
                    // Rysowanie zewnętrznego obramowania tarczy (w pikselach):
                    val gruboscObramowaniaTarczy = 12f
                    drawCircle(
                        color = Color.Gray,
                        radius = promienTarczy + gruboscObramowaniaTarczy,
                        center = Offset(centerX, centerY),
                        style = Stroke(width = gruboscObramowaniaTarczy)
                    )
                }
```

```
}
// Rysujemy logo zegarka
drawIntoCanvas { canvas ->
    val rysik = Paint().asFrameworkPaint() // Tworzymy obiekt rysik klasy Paint
    val logoZegarka = "Epos"
    rysik.textSize = 50.sp.toPx() / density // Density to ilość pikseli na cal
    rysik.color = Color.Black.toArgb()
    rysik.isFakeBoldText = true
    rysik.letterSpacing = 0.3f
    rysik.setShadowLayer( // Warstwa do symulacji cienia
        8f, // radius, czyli rozproszenie cienia
        4f, // dx, czyli przesunięcie na osi X
        4f, // dy, czyli przesunięcie na osi Y
        Color.Gray.toArgb() // kolor cienia
    )
    // Rozmiar tekstu w pikselach i kalkulacja jego pozycji:
    val szerokoscTekstu = rysik.measureText(logoZegarka)
    val wysokoscTekstu = rysik.fontMetrics.bottom - rysik.fontMetrics.top
    val pozycjaLogoX = (size.width - szerokoscTekstu) / 2
    val pozycjaLogoY = (size.height + wysokoscTekstu) / 4
    canvas.nativeCanvas.drawText(logoZegarka, pozycjaLogoX, pozycjaLogoY, rysik)
}

// Rysujemy stopkę na tarczy (Swiss Made):
drawIntoCanvas { canvas ->
    val rysik = Paint().asFrameworkPaint() // Tworzymy obiekt rysik
    val tekstStopki = "Swiss Made"
    rysik.textSize = 20.sp.toPx() / density // Jakie DPI ma napis
    rysik.color = Color.Black.toArgb()
    rysik.letterSpacing = 0.3f
    // Rozmiar tekstu w pikselach i kalkulacja jego pozycji:
    val szerokoscTekstu = rysik.measureText(tekstStopki)
    val pozycjaStopkiX = (size.width - szerokoscTekstu) / 2
    val pozycjaStopkiY = (size.height / 2) * 1.7f // Przesunięcie napisu w dół
    canvas.nativeCanvas.drawText(tekstStopki, pozycjaStopkiX, pozycjaStopkiY,
     rysik)
}

// Ilość indeksów na tarczy: 12.
repeat(12) { hour ->
    val indeksyKat = Math.PI / 6 * hour // Odległość kątowa między indeksami
    val poczatekIndeksuX =
        centerX + cos(indeksyKat) * (promienTarczy - 54) // Kąt nachylenia indeksów
          // (wewnętrzny), długość indeksów od wewnątrz
    val poczatekIndeksuY =
        centerY + sin(indeksyKat) * (promienTarczy - 54) // Kąt nachylenia indeksów
    // (zewnętrzny), długość indeksów od wewnątrz
    val koniecIndeksuX =
```

```
            centerX + cos(indeksyKat) * (promienTarczy - 12) // Odległość indeksów od
                // krawędzi tarczy
        val koniecIndeksuY =
            centerY + sin(indeksyKat) * (promienTarczy - 12) // Odległość indeksów od
            // krawędzi tarczy
        // Obrót o 90 stopni w lewo (ma to znaczenie, gdybyśmy chcieli dodać liczby do
        // indeksów).
        // Atrybut pivot to punkt obrotu:
        rotate(degrees = -90f, pivot = Offset.Zero) {
            // Przesunięcie punktów po obróceniu
            translate(left = -size.height, top = 0f) {
                // Rysowanie powyżej zdefiniowanych indeksów na tarczy:
                drawIntoCanvas { canvas ->
                    val rysik = Paint().asFrameworkPaint() // Tworzymy obiekt rysik
                    // Ustawienie pseudo cienia (przesunięcie dodatkowej warstwy):
                    rysik.setShadowLayer(8f, 4f, 4f, Color.Gray.toArgb())
                    // rysik.strokeCap = Cap.ROUND // Indeksy zaokrąglone
                    canvas.nativeCanvas.drawLine(
                        poczatekIndeksuX.toFloat(),
                        poczatekIndeksuY.toFloat(),
                        koniecIndeksuX.toFloat(),
                        koniecIndeksuY.toFloat(),
                        rysik.apply {
                            color = Color.Black.toArgb()
                            strokeWidth = 6.dp.toPx()
                            // Wyłączenie antyaliasingu dla ostrzejszego efektu cienia
                            isAntiAlias = false
                        }
                    )
                }
            }
        }
    }

    // Poniżej, dzielimy modulo przez 12, aby godzina 13, 14, 15... wskazywała na 1, 2, 3.
    val godzina = (aktualnyCzas.get(Calendar.HOUR_OF_DAY) % 12)
    val minuta = aktualnyCzas.get(Calendar.MINUTE)
    val sekunda = aktualnyCzas.get(Calendar.SECOND)
    val pozycjaKatowaGodziny = Math.PI / 6 * (godzina + minuta / 60.0)
    val pozycjaKatowaMinuty = Math.PI / 30 * (minuta + sekunda / 60.0)
    val pozycjaKatowaSekundy = Math.PI / 30 * sekunda
    // Wskazówki, ich długość i grubość. Wywołania przekazują do funkcji argumenty.
    // Warto wiedzieć: Każda godzina to 30 stopni kątowych.
    // Angielska nazwa wskazówki: hand.


    // WSKAZÓWKA GODZINOWA:
    rysujWskazowke(
```

```
    centerX,
    centerY,
    promienTarczy * 0.5f, // Długość wskazówki oparta o promień tarczy
    pozycjaKatowaGodziny,
    10.dp, // Grubość wskazówki
    Color.Black,
    android.graphics.Paint.Cap.ROUND // Zaokrąglone wskazówka
)
// WSKAZÓWKA MINUTOWA:
rysujWskazowke(
    centerX,
    centerY,
    promienTarczy * 0.86f,  // Długość wskazówki oparta o promień tarczy
    pozycjaKatowaMinuty,-
    8.dp, // Grubość wskazówki
    Color.Black,
    android.graphics.Paint.Cap.ROUND // Zaokrąglona wskazówka
)
// SEKUNDNIK:
rysujWskazowke(
    centerX,
    centerY,
    promienTarczy * 0.95f, // Długość sekundnika oparta o promień tarczy
    pozycjaKatowaSekundy,
    2.dp, // Grubość sekundnika
    Color.Red,
    android.graphics.Paint.Cap.ROUND // Zaokrąglony sekundnik
)

// Obrót centralnych kropek o 90 stopni w lewo
// (może mieć znaczenie w przypadku artystycznego kształtu wskazówek)
rotate(degrees = -90f, pivot = Offset.Zero) {
    // Przesunięcie punktów po obróceniu
    translate(left = -size.height, top = 0f) {
        // Kropki na środku tarczy:
        drawCircle(
            color = Color.Red,
            radius = 20f,
            center = Offset(centerX, centerY),
            style = Fill
        )
        drawCircle(
            color = Color.DarkGray,
            radius = 10f,
            center = Offset(centerX, centerY),
            style = Fill
        )
    }
```

```
            }
        } // Canvas
    } // Box
} // Column
}

// Funkcja zewnętrzna do rysowania wskazówek, przyjmuje argumenty:
fun DrawScope.rysujWskazowke(
    przekazaneCenterX: Float,
    przekazaneCenterY: Float,
    przekazanaDlugoscWskazowki: Float,
    przekazanaPozycjaKatowaWskazowki: Double,
    przekazanaSzerokoscWskazowki: Dp,
    przekazanyKolorWskazowki: Color,
    przekazaneZaokraglenieWskazowki: android.graphics.Paint.Cap
) {
    val koniecWskazowkiX = przekazaneCenterX + cos(przekazanaPozycjaKatowaWskazowki) *
    przekazanaDlugoscWskazowki
    val koniecWskazowkiY = przekazaneCenterY + sin(przekazanaPozycjaKatowaWskazowki) *
    przekazanaDlugoscWskazowki
    // Obrót wskazówek o 90 stopni w lewo,
    // gdyż domyślnie godzina 12 jest na osi X (czyli na godzinie 3):
    rotate(degrees = -90f, pivot = Offset.Zero) {
        // Przesunięcie punktów po obróceniu
        translate(left = -size.height, top = 0f) {
            drawIntoCanvas { canvas ->
                val rysik = Paint().asFrameworkPaint() // Tworzymy obiekt rysik
                // Ustawienie pseudo cienia (przesunięcie dodatkowej warstwy).
                // Cień powinien mieć te same wartości, co przy innych elementach.
                rysik.setShadowLayer(8f, 4f, 4f, Color.Gray.toArgb())
                rysik.color = przekazanyKolorWskazowki.toArgb()
                rysik.strokeCap = przekazaneZaokraglenieWskazowki
                canvas.nativeCanvas.drawLine(
                    przekazaneCenterX,
                    przekazaneCenterY,
                    koniecWskazowkiX.toFloat(),
                    koniecWskazowkiY.toFloat(),
                    rysik.apply {
                        strokeWidth = przekazanaSzerokoscWskazowki.toPx()
                        // Wyłączenie antyaliasingu dla ostrzejszego efektu cienia
                        isAntiAlias = false
                    }
                )
            }
        }
    }
}
```

Ostatnia aktualizacja: 29 lutego 2024.